DESCRIPTION

INFORMATION PROCESSING APPARATUS, INTERRUPT PROCESS CONTROL METHOD, AND COMPUTER PROGRAM

Technical Field

[0001]

The present invention relates to an information processing apparatus, an interrupt process control method, and a computer program. More specifically, the present invention relates to an information processing apparatus, an interrupt control method, and a computer program for setting an main operating system (OS) for controlling an interrupt process in a system including a plurality of OS's in order to decrease interrupt mask time of the entire system, improve interrupt response, and achieve efficient data processing.

Background Art

[0002]

In a multi operating system (OS) having a plurality of OS's in a single system, each OS can execute respective process and hardware common to the system, such as a central processing unit (CPU) and a memory is successively switched in time sequence.

[0003]

Scheduling of processes (tasks) of a plurality of OS's

is executed by a partition management software program, for example. If an OS(α) and an OS(β) coexist in a single system with the process of OS(α) being a partition A and the process of OS(β) being a partition B, the partition management software program determines the scheduling of the partition A and the partition B, and executes the process of the OS's with the hardware resources allocated based on the determined scheduling.

[0004]

Patent Document 1 discloses a task management technique of a multi OS system. According to the disclosure, tasks to be executed by a plurality of OS's are scheduled with a priority placed on a process having urgency.

[0005]

An entity processing data is set up as a partition. More specifically, a logical partition is set up as an entity that shares resources in a system. A variety of resources such as use time of the physical processor, virtual address space, and memory space are allocated to the logical partition. The process is then performed using the allocated resources. A logical processor corresponding to any physical processor is set up in the logical partition, and data processing is performed based on the logical processor. The logical processor does not always correspond to the physical processor on a one-to-one correspondence.

For example, a single logical processor can correspond to a plurality of physical processors, and a plurality of logical processors can correspond to a single physical processor.

[0006]

If a plurality of processes are performed in parallel using the logical processor, the physical processor is used by scheduling the plurality of logical processors. More specifically, the plurality of logical processors uses the physical processor in a time sharing manner.

[0007]

In the multi OS, hardware resources as physically available processors are limited. A physical processor, currently used by one OS, cannot be used by another OS. A duration throughout which the processor cannot be used by the other processor is referred to as an interrupt mask period. As the interrupt mask period is prolonged, the efficiency of the entire system is decreased.

[Patent Document 1] Japanese Unexamined Patent Application Publication No. 2003-345612

Disclosure of Invention

[Problems to Be Solved by the Invention]

[0008]

It is thus desirable to provide an information processing apparatus, an interrupt control method, and a computer program for setting an main OS for controlling an

interrupt process in a system including a plurality of OS's in order to decrease interrupt mask time of the entire system, improve interrupt response, and achieve efficient data processing.

[Means for Solving the Problems]

[0009]

In accordance with one aspect of the present invention, an information processing apparatus processes data for a plurality of OS's. The plurality of OS's includes a main OS controlling an interrupt process and a sub OS. The main OS stores status information as to whether the sub OS is in an interrupt enabled state or an interrupt disabled state, and controls the interrupt process to perform one of an interrupt process execution and an interrupt process reserve in response to the generation of the interrupt based on the status information.

[0010]

In the information processing apparatus of one embodiment of the present invention, the main OS stores interrupt process status information as to whether the interrupt process is in progress or in reserve, and resumes the interrupt process execution in response to the transition of the sub OS between the interrupt enabled state and the interrupt disabled state.

[0011]

In the information processing apparatus of one embodiment of the present invention, the sub OS notifies the main OS of the status information as to whether the sub OS is in the interrupt enabled state or the interrupt disabled state, and the main OS updates the status information of the sub OS in response to the notification from the sub OS.

[0012]

In the information processing apparatus of one embodiment of the present invention, the main OS stores priority information of the interrupt process, and performs the interrupt process responsive to the priority information.

[0013]

In the information processing apparatus of one embodiment of the present invention, the main OS performs status management based on a status table containing the status information of the sub OS and the interrupt process status information as to whether the interrupt process is in progress or in reserve. If an interrupt intended for the sub OS is generated and the main OS determines based on the status table that the sub OS is in the interrupt disabled state, the main OS registers the interrupt in the status table as a reserved interrupt. If an interrupt intended for the sub OS is generated and the main OS determines based on the status table that the sub OS is in the interrupt enabled state, the main OS performs interrupt control depending on

whether the OS operating on a processor is either the main OS or the sub OS in a manner such that

(a) if the main OS is in operation, the main OS

(a1) executes the interrupt process in response to a high priority interrupt, or

(a2) reserves the interrupt process in response to a low priority interrupt, and that

(b) if the sub OS is in operation, the sub OS executes the interrupt process regardless of the priority level of the interrupt.

[0014]

In the information processing apparatus of one embodiment of the present invention, the main OS performs status management based on a status table containing the status information of the sub OS and the interrupt process status information as to whether the interrupt process is in progress or in reserve. If an interrupt intended for the main OS is generated, the main OS performs interrupt control depending on whether the OS operating on a processor is either the main OS or the sub OS in a manner such that

(a) if the main OS is in operation, the main OS executes the interrupt process regardless of the priority level of the interrupt, and that

(b) if the sub OS is in operation, the sub OS

(b1) executes the interrupt process in response to a

high priority interrupt, or

(b2) reserves the interrupt process in response to a low priority interrupt.

[0015]

In accordance with a second aspect of the present invention, an interrupt process control method for performing data processing on a plurality of OS's, includes steps of receiving, from a sub OS other than an main OS, status information as to whether the sub OS is in an interrupt enabled state or an interrupt disabled state, detecting the generation of an interrupt, and controlling an interrupt process to perform one of an interrupt process execution and an interrupt process reserve in response to the generation of the interrupt based on the status information.

[0016]

In the interrupt process control method of one embodiment of the present invention, the main OS stores interrupt process status information as to whether the interrupt process is in progress or in reserve, and resumes the interrupt process execution in response to the transition of the sub OS between the interrupt enabled state and the interrupt disabled state.

[0017]

The interrupt process control method of one embodiment

of the present invention may further include steps of notifying the main OS of the status information as to whether the sub OS is in the interrupt enabled state or the interrupt disabled state, and updating the status information of the sub OS in response to the notification from the sub OS.

[0018]

In the interrupt process control method of one embodiment of the present invention, the main OS stores priority information of the interrupt process, and performs the interrupt process responsive to the priority information.

[0019]

In the interrupt process control method of one embodiment of the present invention, the main OS performs status management based on a status table containing the status information of the sub OS and the interrupt process status information as to whether the interrupt process is in progress or in reserve. If an interrupt intended for the sub OS is generated and the main OS determines based on the status table that the sub OS is in the interrupt disabled state, the main OS registers the interrupt in the status table as a reserved interrupt. If an interrupt intended for the sub OS is generated and the main OS determines based on the status table that the sub OS is in the interrupt enabled state, the main OS performs interrupt control depending on

whether the OS operating on a processor is either the main OS or the sub OS in a manner such that

(a) if the main OS is in operation, the main OS

(a1) executes the interrupt process in response to a high priority interrupt, or

(a2) reserves the interrupt process in response to a low priority interrupt, and that

(b) if the sub OS is in operation, the sub OS executes the interrupt process regardless of the priority level of the interrupt.

[0020]

In the interrupt process control method of one embodiment of the present invention, the main OS performs status management based on a status table containing the status information of the sub OS and the interrupt process status information as to whether the interrupt process is in progress or in reserve. If an interrupt intended for the main OS is generated, the main OS performs interrupt control depending on whether the OS operating on a processor is either the main OS or the sub OS in a manner such that

(a) if the main OS is in operation, the main OS executes the interrupt process regardless of the priority level of the interrupt, and that

(b) if the sub OS is in operation, the sub OS

(b1) executes the interrupt process in response to a

high priority interrupt, or

(b2) reserves the interrupt process in response to a low priority interrupt.

[0021]

In accordance with a third aspect of the present invention, a computer program for performing an interrupt process control in data processing on a plurality of OS's, includes steps of receiving, from a sub OS other than an main OS, status information as to whether the sub OS is in an interrupt enabled state or an interrupt disabled state, detecting the generation of an interrupt, and controlling an interrupt process to perform one of an interrupt process execution and an interrupt process reserve in response to the generation of the interrupt based on the status information.

[0022]

The computer program of one embodiment of the present invention is provided, to a general-purpose computer system executing a variety of program code, in a computer-readable storage medium, such as a CD, an FD, or an MO, or a communication medium such as network. By providing the computer program in a computer readable manner, the computer system performs process responsive to the computer program.

[0023]

These and other features, and advantages of the present

invention will become obvious from the following description of the present invention and the accompanying drawings. In the context of the description of the present invention, the system refers to a logical set of a plurality of apparatuses, and is not limited to an apparatus that houses elements within the same casing.

[Advantages]

[0024]

In accordance with embodiments of the present invention, the main OS executing the interrupt process is set in the system in which a plurality of OS's concurrently run. With the main OS performing interrupt control process, the mask time of the entire system is reduced, interrupt response is improved, and data processing is efficiently performed.

[0025]

In accordance with embodiments of the present invention, the main OS executing the interrupt process control is set, and the right to set interrupt mask is not handed over to the sub OS other than the main OS. The sub OS notifies the main OS whether the sub OS is in the interrupt enabled state or the interrupt disabled state. The main OS controls interrupt mask of the sub OS. This arrangement prevents the sub OS from reserving a required interrupt process due to own mask control. In accordance with the intention of the main OS, all interrupt process is controlled. Required

interrupt processes are thus performed with priority.

[0026]

In accordance with embodiments of the present invention, a sub OS interrupt vector management unit is set in the main OS. The main OS generally manages an interrupt vector area of the sub OS. Unlike interrupt vector management individually performed by OS's, the interrupt vector is shared, and memory area is saved.

Brief Description of the Drawings

[0027]

[Fig. 1] Fig. 1 is a block diagram illustrating an information processing apparatus of one embodiment of the present invention.

[Fig. 2] Fig. 2 illustrates the structure of a processor module.

[Fig. 3] Fig. 3 illustrates operating systems of the information processing apparatus in accordance with one embodiment of the present invention.

[Fig. 4] Fig. 4 illustrates an allocation process of allocating a logical processor to a physical processor in a time sharing manner.

[Fig. 5] Fig. 5 is a functional diagram illustrating a main OS of the information processing apparatus in accordance with one embodiment of the present invention.

[Fig. 6] Fig. 6 illustrates a status table listing OS

status information and interrupt process status information managed by the main OS in one embodiment of the present invention.

[Fig. 7] Fig. 7 illustrates the status of the sub OS and a process sequence responsive to the generation of an interrupt.

[Fig. 8] Fig. 8 illustrates a list of relationship of OS's executing a process using a processor (OS's in operation), priority level of the interrupt (high and low priority levels), an interrupt destination OS, and an interrupt grant state of the interrupt destination OS.

Best Mode for Carrying Out the Invention

[0028]

An information processing apparatus, an interrupt process control method, and a computer program in accordance with embodiments of the present invention are described below with reference to the drawings.

[0029]

The hardware structure of the information processing apparatus of one embodiment of the present invention is described below with reference to Fig. 1. A processor module 101 includes a plurality of processing units, and processes data in accordance with a variety of programs stored in a read-only memory (ROM) 104 and an HDD 123, including operating systems (OS's) and application programs

running on the OS. The processor module 101 will be described later with reference to Fig. 2.

[0030]

In response to a command input via the processor module 101, a graphic engine 102 generates data to be displayed on a screen of a display forming an output unit 122, for example, performs a 3D graphic drawing process. A main memory (DRAM) 103 stores the program executed by the processor module 101 and parameters that vary in the course of execution of the program. These elements are interconnected via a host bus 111 including a CPU bus.

[0031]

The host bus 111 is connected to an external bus 112, such as a PCI (peripheral component interconnect/interface) bus via a bridge 105. The bridge 105 controls data inputting and outputting between the host bus 111, the external bus 112, a controller 106, a memory card 107, and other devices.

[0032]

An input unit 121 inputs information to an input device, such as a keyboard and a pointing device, operated by a user. An output unit 122 includes an image output unit, such as one of a liquid-crystal display and a CRT (cathode ray tube), and an audio output device such as a loudspeaker.

[0033]

The HDD (hard disk drive) 123 drives a hard disk loaded therewithin, thereby recording or playing back a program to be executed by the processor module 101 and information.

[0034]

A drive 124 reads data and programs stored in a loaded removable recording medium 127, such as a magnetic disk, an optical disk, a magneto-optic disk, a semiconductor memory, or the like, and supplies the data and the programs to a main memory (DRAM) 103 via an interface 113, the external bus 112, the bridge 105, and the host bus 111.

[0035]

A connection port 125 connects to an external device 128, and may include a USB, an IEEE 1394 bus, or the like. The connection port 125 is connected to the processor module 101 via the interface 113, the external bus 112, the bridge 105, and the host bus 111. A communication unit 126, connected to a network, transmits data supplied from the HDD 123 or the like, and receives data from the outside.

[0036]

The structure of the processor module is described below with reference to Fig. 2. As shown, a processor module 200 includes a main processor group 201 including a plurality of main processor units, and a plurality of sub-processor groups 202 - 20n, each including a plurality of sub-processor units. Each group further includes a memory

controller and a secondary cache. The processor groups 201 - 20n, each including eight processor units, for example, are connected via one of a cross-bar architecture and a packet exchange network. In response to a command of the main processor of the main processor group 201, at least one sub-processor in the plurality of sub-processor groups 202 - 20n is selected to perform a predetermined program.

[0037]

The structure of the processor module is described. The memory-flow controller in each processor group controls data inputting and data outputting to the main memory 103 of Fig. 1. The secondary cache serves as a memory area for process data in each processor group.

[0038]

The operating systems (OS's) of the information processing apparatus of one embodiment of the present invention are described below with reference to Fig. 3. The multi-OS information processing apparatus has a plurality of OS's arranged in a logical layered structure as shown in Fig. 3.

[0039]

As shown in Fig. 3, a main OS 301 is arranged at a lower layer. A plurality of sub OS's 302, 303, and 304 are arranged at upper layers. Guest sub OS's 302 and 303 and sub a system control OS 304 are set as sub OS's. Together

with the system control OS 304, the main OS 301 forms a logical partition as an execution unit of each process executed by the processor module 101 discussed with reference to Figs. 1 and 2, and allocates system hardware resources (for example, main processors, sub-processors, memories, and devices, as computing resources) to each logical partition.

[0040]

The guest OS's 302 and 303 as the sub OS's are a gaming OS, Windows®, Linux®, etc, and operate under the control of the main OS 301. Although only two guest OS's 302 and 303 are shown in Fig. 3, the number of guest OS's is not limited to two.

[0041]

The guest OS's 302 and 303 operate within the logical partitions set by the main OS 301 and the system control OS 304. The guest OS's 302 and 303 process a variety of data using hardware resources such as main processors, sub-processors, memories, and device, each allocated to the logical partition.

[0042]

The guest OS(a) 302 uses the hardware devices including a main processor, a sub-processor, a memory, and a device allocated to the logical partition 2 set up by the control OS 301 and the system control OS 304, thereby executing an

application program 305 corresponding to the guest OS(a) 302. The guest OS(b) 303 uses the hardware resources including a main processor, a sub-processor, a memory, and a device allocated to a logical partition "n", thereby executing an application program 306 corresponding to the guest OS(b) 303. The main OS 301 provides a guest OS programming interface required to execute the guest OS.

[0043]

The system control OS 304 as one of the sub OS's generates a system control program 307 containing logical partition management program, and performs operation control responsive to the system control program 307 together with the main OS 301. The system control program 307 controls system policy using a system control program programming interface. The application program 306 is supplied with the system control program programming interface by the control OS 301. For example, the system control program 307 permits flexible customization, for example, setting an upper limit on resource allocation.

[0044]

The system control program 307 controls the behavior of the system using the system control program programming interface. For example, the system control program 307 produces a new logical partition, and starts up a new guest OS at the logical partition. In a system where a plurality

of guest OS's are operating, the guest OS's are initiated in the order programmed in the system control program 307. The system control program 307 can receive and examine a resource allocation request issued from the guest OS before being received by the main OS 301, modify the system policy, and deny the request itself. In this way, no particular guest OS monopolizes the resources. A program into which the system policy is implemented is the system control program.

[0045]

The main OS 301 allocates a particular logical partition (for example, the logical partition 1) to the system control OS 304. The main OS 301 operates in a hypervisor mode. The guest OS operates in a supervisor mode. The system control OS 304 and the application program operate in a problem mode (user mode).

[0046]

The logical partition is an entity receiving a resource allocation in the system. For example, the main memory 103 is partitioned into several areas (see Fig. 1), and each logical partition is granted the right to use the respective area. The types of resources allocated to the logical partitions are listed below.

a) Physical processor unit usage time

b) Virtual address space

c) Memory accessible by program operating in a logical partition

d) Memory used by the control OS to manage the logical partition

e) Event port

f) Right to use device

g) Cache partition

h) Right to use bus

[0047]

As previously discussed, each OS operates within the logical partition. The OS monopolizes the resources allocated to the logical partition to process a variety of data. In many cases, one partition is produced for a guest OS, on a per guest OS basis, functioning on the system. Each logical partition is assigned a unique identifier. The system control OS 304 manages the system control program generated as logical partition management information by associating the system control program to the identifier.

[0048]

The logical partition is generated by the main OS 301 and the system control OS 304. Immediately after production, the logical partition has no resources, and with no limitation set on available resources. The logical partition takes one of two states, an active state and an end state. The logical partition immediately after

production takes an active state. The logical partition is transitioned into the end state in response to a request of a guest OS operating in the logical partition, and stops all logical processors allocated to the logical partition.

[0049]

The logical processor is the one allocated to the logical partition, and corresponds to any physical processor, namely, a processor within a processor group of Fig. 2. The logical processor and the physical processor are not always related to each other in one-to-one correspondence. A single logical processor can correspond to a plurality of physical processors. Alternatively, a plurality of logical processors can correspond to a single physical processor. The correspondence between the logical processor and the physical processor is determined by the main OS 301.

[0050]

The main OS 301 has a function to limit the amount of resources available to each logical partition. Limitation can be set to the amount of use of resources the guest OS 302 and the guest OS 303 can allocate and release without communicating with the system control OS 304.

[0051]

The main OS provides the logical partition with a logical sub-processor in an abstract form of a physical sub-processor (as a computing resource). As previously

discussed, the physical sub-processor is not related to the logical sub-processor in one-to-one correspondence, and it is not a requirement that the physical sub-processors be identical in number to the logical sub-processors. As necessary, the control OS can thus cause a single physical sub-processor to correspond to a plurality of logical sub-processors.

[0052]

If the number of the logical sub-processors is larger than the number of physical sub-processors, the control OS uses the physical sub-processors in a time sharing manner. The logical sub-processor can repeatedly stop and then resume operation. The sub OS can monitor such changes.

[0053]

An entity processing data is set up as a partition. More specifically, a logical partition is set up as an entity that shares resources in a system. A variety of resources such as use time of the physical processor, virtual address space, and memory space are allocated to the logical partition. The process is then performed using the allocated resources. A logical processor corresponding to any physical processor is set up in the logical partition, and data processing is performed based on the logical processor. The logical processor does not always correspond to the physical processor on a one-to-one correspondence

basis. For example, a single logical processor can correspond to a plurality of physical processors, and a plurality of logical processors can correspond to a single physical processor.

[0054]

If a plurality of processes are performed in parallel using the logical processor, the physical processor is scheduled by the plurality of logical processors. More specifically, the plurality of logical processors uses the physical processor in a time sharing manner.

[0055]

Referring to Fig. 4, how to use the physical processor in a time sharing manner is discussed. As shown in Fig. 4(a), a single logical processor corresponding to one of the main OS and the sub OS is allocated to a single physical processor. A logical processor (a) monopolizes a physical processor (1), and a logical processor (b) monopolizes a physical processor 2.

[0056]

As shown in Fig. 4(b), a plurality of logical processors, assigned to a single physical processor, perform processes in a time sharing manner. The physical processor 1 is shared in time in the order of the logical processors c → a → c → a → b → c → b. Each logical processor corresponding to the process of one of the main OS and the

sub OS is performed. The physical processor 2 is shared in time in the order of the logical processors b → d → b → d → c → d → a. Each logical processor corresponding to the process of one of the main OS and the sub OS is performed.

[0057]

The information processing apparatus of one embodiment of the present invention is a multi-OS system using a plurality of OS's. As previously discussed with reference to Fig. 3, the plurality of OS's are divided into a single main OS and remaining sub OS's. The function and interrupt process control of the main OS and the sub OS's of the information processing apparatus of one embodiment of the present invention are described below.

[0058]

In the interrupt process, a device such as an input and output device or a system clock, interrupts asynchronously a CPU. To accept an interrupt and perform an interrupt process, the CPU as a physical processor suspends a current process in the middle thereof, performs the interrupt process, and then resume the original process after completion of the interrupt process. In the interrupt process, hardware status information corresponding to the reserved process is produced as a context table, and stored in memory as necessary. Subsequent to the completion of the interrupt process, the CPU restores hardware state by

recovering context, and resumes the process from reserve.

[0059]

The functions of in the interrupt process control of the main OS and the sub OS are listed as follows:

(1) The main OS manages competing resources such as setting interrupt mask and interrupt vector.

(2) The main OS records status information of OS's processing data, including the main OS.

(3) Instead of directly controlling an interrupt mask register, the sub OS notifies the main OS whether the sub OS is in an interrupt enabled state or an interrupt disabled state.

(4) The main OS receives all interrupt requests, and transfers a required interrupt only when the sub OS is interruptable and operative.

[0060]

With these functions,

(1) all interrupt masks are emptied for a period throughout which all interrupts are masked from the sub OS.

(2) Since the interrupt vector is shared, an integrated image of the main OS and the sub OS is produced to reduce used memory.

[0061]

The functions of the main OS are described below with reference to Fig. 5. Fig. 5 is a block diagram illustrating

the function of a main OS 510 in the interrupt process control.

[0062]

The main OS 510 includes an interrupt priority manager 511, a sub OS interrupt manager 512, a sub OS interrupt vector manager 513, an interrupt factor registration manager 514, an executing OS state manager 515, an interrupt deliverer 516, an executing OS switch controller 517, an interrupt process end notifier 518, and an interrupt reserve controller 519. The process of each element is described below.

[0063]

The interrupt priority manager 511 performs interrupt management based on the priority corresponding to an interrupt factor. The interrupt processes include an interrupt process intended for the main OS and an interrupt process intended for the sub OS, and each interrupt is prioritized. In the prioritization, each interrupt process is prioritized with a two-level priority of low level and high level, or three-level or more multi-level priority. The interrupt priority manager 511 determines a process corresponding to an interrupt 530 caused in accordance with the priority set in response to the interrupt factor.

[0064]

The sub OS interrupt manager 512 manages the interrupt

enabled state and the interrupt disabled state of the sub OS. As previously discussed, the main OS 510 is notified by the sub OS 520 whether the sub OS is in the interrupt enabled state or the interrupt disabled state. In response to the notification from the sub OS 520, the sub OS interrupt manager 512 in the main OS 510 stores management information as to whether each sub OS is in the interrupt enabled state or the interrupt disabled state.

[0065]

The sub OS interrupt manager 512 registers the state of each sub OS onto a status table listing the state as to whether each sub OS is in the interrupt enabled state or the interrupt disabled state. For example, the status table contains an interrupt mask register that stores the interrupt enabled state as (0) and the interrupt disabled state as (1). The sub OS interrupt manager 512 manages the state of each sub OS based on the interrupt mask register. If an interrupt request occurs with the sub OS in the interrupt disabled state, namely, in a masked state, the interrupt request must wait. With the sub OS in the interrupt disabled state, namely, in a mask released state, the interrupt process responsive to the interrupt request is permitted.

[0066]

Fig. 5 illustrates a single sub OS 520. As previously

discussed with reference to Fig. 3, the information processing apparatus can accommodate a plurality of sub OS's. The sub OS interrupt manager 512 in the main OS 510 stores the management information of each of the sub OS's as to whether the sub OS is in the interrupt enabled state or the interrupt disabled state, and performs interrupt process control in response to the state of each sub OS.

[0067]

Fig. 6 illustrates the status table stored as the management information of the sub OS interrupt manager 512. As shown in Fig. 6, the status table lists, in a related form, data of the state of each OS, namely, the status information as to whether the sub OS is in the interrupt enabled state or the interrupt disabled state, information relating to a reserved interrupt, and information relating to the interrupt process in progress. Upon receiving the status notification from each sub OS, the sub OS interrupt manager 512 updates the status table in response to the received information. In response to the occurrence, reserve, delivery, and completion of an interrupt process, the sub OS interrupt manager 512 registers in the status table the status as to whether each interrupt is reserved or in progress. The sub OS interrupt manager 512 deletes an interrupt from the status table if the corresponding interrupt process has been completed.

[0068]

The sub OS interrupt vector manager 513 manages an interrupt vector area in each sub OS. The interrupt vector is a memory area table defined by the interrupt factor, and is composed of a start address of an interrupt process routine. A processor receiving an interrupt examines the address of an interrupt handler from the memory area, and jumps to the address to start the interrupt process. The sub OS interrupt vector manager 513 manages the interrupt vector area of the sub OS.

[0069]

The interrupt factor registration manager 514 stores and manages registration information of the interrupt factor intended for the main OS and the interrupt factor intended for the sub OS. The interrupt factors of the main OS and the sub OS can be modified by an I/O device used by the OS. The interrupt factor registration manager 514 registers and deletes the interrupt factors to each OS. The interrupt factors include an interrupt process from an input unit, such as a keyboard or a mouse, and data input from a network interface.

[0070]

The executing OS state manager 515 records and manages information of an OS executing data processing on the physical processor.

The interrupt deliverer 516 performs an interrupt delivery process, namely, determines which processor or processor group is to be notified of each interrupt request to perform the corresponding interrupt process.

[0071]

The executing OS switch controller 517 controls switching a variety of data processings (including tasks) to be performed by the processor. As previously discussed, the logical processors are allocated to the physical processor. At least one processor is shared in time by the OS's to process data. The OS switch controller 517 switches the OS's in accordance with the processor use scheduling of the OS's.

[0072]

The interrupt process end notifier 518 monitors execution status of the interrupt process in response to the interrupt request, and notifies the OS, having executed the process suspended by the interrupt process, of the end of the interrupt process. When the interrupt process is completed, the context corresponding to the suspended process is restored, the hardware state of the suspended process is restored, and then the original process resumes.

[0073]

The interrupt reserve controller 519 performs an interrupt reserve control process. As previously discussed,

the interrupt request occurring in the interrupt disabled state, namely, in the masked state must wait until masking is released. The interrupt reserve controller 519 manages a reserved interrupt on standby state. The interrupt reserve controller 519 updates the status table having interrupt reserve information set therewithin and notifies the sub OS 520 of the reserved interrupt if the reserved interrupt is registered in the status table.

[0074]

The interrupt process control and state transition performed by the main OS in response to the two states of the sub OS, namely, the "interrupt enabled state" and the "interrupt disabled state", is described below with reference to Fig. 7.

[0075]

As shown in Fig. 7, steps S101 - S104 show fours states of a single sub OS in the information processing apparatus as follows:

State S101: the sub OS starts.

State S102: the sub OS is initialized.

State S103: the sub OS is set to be in the interrupt disabled state.

State S104: the sub OS is set to be in the interrupt enabled state.

[0076]

With the sub OS being in one of:

state S103: interrupt disabled state and

state S104: interrupt enabled state, one of the following events occurs:

event I201: interrupt to the main OS, or

event I202: interrupt to the sub OS

[0077]

Process sequences 1 - 9 responsive to each of transitions between the four states of S101 - S104 of the sub OS are described below. One of the process sequences 1 through 9 is performed when one of the event I201, namely, the interrupt for the main OS, and the event I202, namely, the interrupt for the sub OS takes place.

[0078]

[Sequence 1]

The sequence 1 is performed in a state transition from

state S101: startup of the sub OS to

state S102: completion of initialization of the sub OS.

During this state transition, the sub OS notifies the main OS of an interrupt factor used by the sub OS. This step is performed as the notification step to the interrupt factor registration manager 514 of Fig. 5.

The interrupt factor registration manager 514 of the main OS examines the interrupt factor notified of by the sub OS, and registers the permitted interrupt factor as a sub OS

interrupt factor.

[0079]

[Sequence 2]

The sequence 2 is performed in response to the state transition from

step S102: completion of initialization of the sub OS to step S103: interrupt disabled state of the sub OS.

During the transitional, the sub OS notifies the main OS that the sub OS is in the interrupt disabled state. This step is performed as the notification step to the sub OS interrupt manager 512 of Fig. 5.

[0080]

Upon receiving the notification from the sub OS that the sub OS is in the interrupt disabled state, the sub OS interrupt manager 512 of the main OS registers the interrupt disabled state of the sub OS in the state table listing the mask state (see Fig. 6). By controlling the interrupt mask register with the interrupt disabled state set as being (0) and the interrupt enabled state set as being (1), the sub OS interrupt manager 512 sets a (mask) state to indicate that the sub OS is in the interrupt disabled state. If an interrupt request is input under this state, that interrupt request must wait.

[0081]

[Sequence 3]

The sequence 3 is performed in response to a state transition from

state S102: completion of initialization of the sub OS to state S104: interrupt enabled state of the sub OS.

During the state transition, the sub OS notifies the main OS that the sub OS is in the interrupt enabled state. The step is performed as the notification step to the sub OS interrupt manager 512 of Fig. 5.

[0082]

Upon receiving from the sub OS the notification that the sub OS is in the interrupt enabled state, the sub OS interrupt manager 512 in the main OS registers the interrupt enabled state of the sub OS in the status table listing the mask state (see Fig. 6). By controlling the interrupt mask register with the interrupt disabled state set as being (0) and the interrupt enabled state set as being (1), the sub OS interrupt manager 512 sets a (mask release) state to indicate that the sub OS is in the interrupt enabled state. If an interrupt request is input under this state, that interrupt request is performed without waiting.

[0083]

The main OS checks the state table for the presence of an interrupt currently reserved. The status table lists OS status information as to whether each OS is in the interrupt enabled state or the interrupt disabled state, and interrupt

reserved state information. The interrupt reserve controller 519 of Fig. 5 writes the interrupt reserved state information onto the status table. If a reserved interrupt is present, the interrupt reserve controller 519 notifies the sub OS of the reserved interrupt.

[0084]

[Sequence 4]

The sequence 4 is performed in response to a state transition from

state S104: sub OS interrupt enabled state to

state S103: sub OS interrupt disabled state.

During the state transition, the sub OS notifies the main OS that the sub OS is transitioned from the interrupt enabled state to the interrupt disabled state. This step is performed as the notification step to the sub OS interrupt manager 512 of Fig. 5.

[0085]

Upon receiving from the sub OS the notification that the sub OS is transitioned from the interrupt enabled state to the interrupt disabled state, the sub OS interrupt manager 512 in the main OS updates the status table (see Fig. 6) listing the mask state, namely, changing state registration information of the sub OS from the interrupt enabled state to the interrupt disabled state. By controlling the interrupt mask register with the interrupt

disabled state set as being (0) and the interrupt enabled state set as being (1), the sub OS interrupt manager 512 sets a (mask) state to indicate that the sub OS is in the interrupt disabled state. If an interrupt request is input under this state, that interrupt request must wait.

[0086]

[Sequence 5]

The sequence 5 is performed in response to a state transition from

state S103: sub OS interrupt disabled state to

state S104: sub OS interrupt enabled state.

During the state transition, the sub OS notifies the main OS that the sub OS is transitioned from the interrupt disabled state to the interrupt enabled state. This step is performed as the notification step to the sub OS interrupt manager 512 of Fig. 5.

[0087]

Upon receiving from the sub OS the notification that the sub OS is transitioned from the interrupt disabled state to the interrupt enabled state, the sub OS interrupt manager 512 in the main OS updates the status table (see Fig. 6) listing the mask state, thereby changing the state registration information of the sub OS from the interrupt disabled state to the interrupt enabled state. By controlling the interrupt mask register with the interrupt

disabled state set as being (0) and the interrupt enabled state set as being (1), the sub OS interrupt manager 512 sets a (mask released) state to indicate that the sub OS is in the interrupt enabled state. If an interrupt request is input under this state, the interrupt process is performed.

[0088]

[Sequence 6]

The sequence 6 is performed when the sub OS interrupt occurs as the event I202 in the sub OS interrupt disabled state of S103. Upon detecting the event I202, namely, the occurrence of the sub OS interrupt, the main OS references the status table to determine whether the sub OS is in the interrupt enabled state or the interrupt disabled state. The sub OS is now in the interrupt disabled state.

After verifying that the sub OS is in the interrupt disabled state, the main OS registers the interrupt as reserved on the status table.

The interrupt process is not currently performed but reserved.

[0089]

[Sequence 7]

The sequence 7 is performed when the sub OS interrupt occurs as the event I202 in the sub OS interrupt enabled state of S104.

[0090]

The process in the sequence 7 becomes different depending on whether the process of the processor is performed by the main OS or the sub OS.

(a) Main OS in operation

The main OS detects the sub OS interrupt as the event I202 while performing the process using the processor. The main OS references of the status table of the sub OS to determine whether the sub OS is in the interrupt enabled state or the interrupt disabled state. The sub OS is in the interrupt enabled state (S104).

After verifying that the sub OS is in the interrupt enabled state, the main OS examines the priority of the interrupt. The interrupt is prioritized with a high priority level or a low priority level. The interrupt process responsive to the high priority level and the low priority level is described below.

[0091]

(a-1) Interrupt process at high priority level

If the generated interrupt factor is at a high priority level, the following processes (1) through (7) are performed. In the following, [Main OS] represents a process performed by the main OS, and [Sub OS] represents a process performed by the sub OS.

(1)[Main OS] The main OS registers the generated interrupt, as being processed, as data of the sub OS

responsive to the interrupt of the status table (Fig. 6).

(2) [Main OS] The main OS switches the application of the processor from the main OS to the sub OS. This switch step is performed as a step of the OS switch controller 517 of Fig. 5.

(3) [Main OS] The main OS delivers the interrupt to the sub OS. This process step is performed as a step of the interrupt deliverer 516 of Fig. 5.

(4) [Sub OS] The sub OS executes the interrupt process responsive to the interrupt.

(5) [Sub OS] Subsequent to the completion of the interrupt process, the sub OS notifies the main OS of the completion of the interrupt process. The interrupt process end notifier receives the end notice from the sub OS.

(6) [Main OS] The main OS deletes from the status table an entry corresponding to the interrupt process registered and completed.

(7) [Main OS] After verifying that no other high priority level interrupt is input, the main OS switches from the sub OS to the main OS. This process step is performed as a step of the OS switch controller 517 of Fig. 5.

[0092]

With the above-described process steps, the sub OS interrupt process having the high priority level is performed with priority. The interrupt process at the low

priority level is described below. If the generated interrupt factor is at a low priority level, the main OS registers the generated interrupt as reserved interrupt information of the sub OS corresponding to the interrupt in the status table (Fig. 6).

At this point of time, the generated interrupt is not executed but must wait.

[0093]

(b) Sub OS in operation

If the sub OS performs the process using the processor, the following process steps are performed. In the following, [Main OS] represents a process performed by the main OS, and [Sub OS] represents a process performed by the sub OS.

(1) [Main OS] Upon detecting the generation of a sub OS interrupt, the main OS references the status table of the sub OS to determine whether the sub OS is in the interrupt enabled state or the interrupt disabled state. In this case, the sub OS is in the interrupt enabled state (S104).

(2) [Main OS] After verifying that the sub OS is in the interrupt enabled state, the main OS registers the generated interrupt, as being processed, as data of the sub OS corresponding to the interrupt in the status table (Fig. 6).

(3) [Main OS] The main OS delivers the interrupt request to the sub OS. This process step is performed as a step of the interrupt deliverer 516 of Fig. 5.

(4) [Sub OS] The sub OS performs the interrupt process.

(5) [Sub OS] Upon completing the interrupt process, the sub OS notifies the main OS of the end of the interrupt process. The interrupt process end notifier 518 of Fig. 5 receives the end notice from the sub OS.

(6) [Main OS] The main OS deletes from the status table an entry corresponding to the interrupt process registered and completed.

[0094]

The sub OS performs the interrupt process responsive to the generated interrupt using the processor. The sub OS, if in the interrupt enabled state, performs the interrupt process without waiting.

[0095]

In summary, the interrupt process is performed in the sequence 7, namely, when the sub OS interrupt occurs with the sub OS in the interrupt enable state.

(A) If the main OS is in operation,

(A-1) the interrupt process is performed in response to a high priority level interrupt, and

(A-2) the interrupt process is reserved in response to a low priority level interrupt.

(B) If the sub OS is in operation, the interrupt process is performed regardless of the priority level of the interrupt.

[0096]

[Sequence 8]

The sequence 8 is performed if the event I201 as the main OS interrupt takes place with the sub OS in the interrupt disabled state of S103.

[0097]

In this case as well, the sequence 8 becomes different depending on whether the process using the processor is performed by the main OS or the sub OS.

(a) Main OS in operation

If the main OS detects the event I201 as a main OS interrupt while the main OS performs a process using the processor, the generated interrupt is delivered to the main OS. The main OS performs the interrupt process.

[0098]

(b) Sub OS in operation

If the main OS detects the event I201 as a main OS interrupt while the sub OS performs a process using the processor, the main OS examines the priority of the generated interrupt. The two priority levels of low and high are used herein. The interrupt processes responsive to a high priority level interrupt and a low priority level interrupt are separately described below.

[0099]

(b-1) High priority level interrupt

The following process steps (1) through (7) are performed if the interrupt factor is at a high priority level. In the following, [Main OS] represents a process performed by the main OS, and [Sub OS] represents a process performed by the sub OS.

(1) [Main OS] The main OS switches the processor applied process from the sub OS to the main OS. The switching step is executed as a step of the OS switch controller 517 of Fig. 5.

(2) [Main OS] The main OS performs the interrupt process.

(3) [Main OS] The main OS switches the processor applied process from the main OS to the sub OS. The switching step is executed as a step of the OS switch controller 517 of Fig. 5.

[0100]

The main OS interrupt process responsive to the high priority level interrupt is thus performed. The main OS interrupt process responsive to the low priority interrupt is described below. If the generated interrupt is at a low priority level, the main OS registers, in the status table (Fig. 6), the generated interrupt as the reserved interrupt information of the main OS.

The generated interrupt is not executed at this point of time but must wait.

[0101]

[Sequence 9]

The sequence 9 is performed if the event I201 as a main OS interrupt takes place with the sub OS in the interrupt enabled state of S104.

[0102]

This process is identical to the process at the above-referenced sequence 8. More specifically, if an interrupt intended for the main OS takes place, the same process is performed regardless of whether the sub OS is in the interrupt enabled state or the interrupt disabled state.

The processes performed in response to the main OS interrupt (sequences 8 and 9) are summarized as below.

(A) The interrupt process is performed regardless of the interrupt priority level if the interrupt occurs while the main OS is in operation.

(B) If an interrupt occurs while the sub OS is in operation, the interrupt process is

(B-1) performed in response to the high priority level interrupt but

(B-2) reserved in response to the low priority level interrupt.

[0103]

Fig. 8 lists OS's executing the processor applied process (OS' in operation), the priority levels (high or

low), interrupt destination OS's, and interrupt enabled and disabled states of the interrupt destination OS's. As listed, 16 scenarios of state setting are possible.

[0104]

In accordance with one embodiment of the present invention, the main OS stores and manages the status information (status table of Fig. 7) of all sub OS's as to whether each sub OS is in the interrupt enabled state or the interrupt disabled state.

[0105]

Responses to the sub OS interrupt process and the main OS interrupt process are summarized as below.

[0106]

[Response to the sub OS interrupt process]

When an interrupt intended for the sub OS takes place, the main OS references the status table. If the sub OS is in the interrupt disabled state, the main OS sets and registers, in the status table, the generated interrupt as being reserved (sequence 6).

If the sub OS is in the interrupt enabled state, the interrupt reserve process or the interrupt process is performed as below depending on whether the OS executing the processor applied process is the main OS or the sub OS (sequence 7).

(A) If the main OS is in operation, the interrupt

process is

(A-1) executed in response to the high priority level interrupt but

(A-2) reserved in response to the low priority level interrupt.

(B) If the sub OS is in operation, the interrupt process is executed regardless of the priority level of the interrupt.

[0107]

[Response to the main OS interrupt process]

When an interrupt intended for the main OS takes place, the interrupt reserve process or the interrupt process is performed as below depending on whether the OS executing the processor applied process is the main OS or the sub OS (sequences 8 and 9).

(A) If the main OS is in operation, the interrupt process is executed regardless of the priority level of the interrupt.

(B) If the sub OS is in operation, the interrupt process is

(B-1) executed in response to the high priority level interrupt but

(B-2) reserved in response to the low priority level interrupt.

[0108]

The main OS controls execution of the interrupt process and the interrupt reserve process based on the status information of all sub OS's, namely, depending on whether each sub OS is in the interrupt enabled state or the interrupt disabled state, status information at the generation of the interrupt, namely, the priority level of the interrupt, and whether the generated interrupt is intended for the main OS or the sub OS.

[0109]

In accordance with embodiments of the present invention, the right to mask interrupts is not handed over to the sub OS. The sub OS notifies the main OS whether the sub OS is in the interrupt enabled state and the interrupt disabled state. Based on the notification, the main OS controls the interrupt masking to the sub OS. This arrangement overcomes the inconvenience that the sub OS reserves a required interrupt process in accordance with the sub OS's own mask control. All interrupt process is thus controlled by the main OS. Since the sub OS interrupt vector manager 513 in the main OS (see Fig. 5) manages the interrupt vector area of the sub OS, the interrupt vector is shared, unlike the case in which interrupt vector management is performed by individual OS's. As previously discussed, the interrupt vector is the table of the memory area defined by the interrupt factor. For example, the interrupt vector is

composed of a start address of an interrupt process routine. A processor receiving an interrupt examines the address of an interrupt handler from the memory area, and jumps to the address to start the interrupt process. The interrupt vector is shared by the main OS and the sub OS, and the required memory area is thus reduced.

[0110]

The present invention has been discussed with reference to the particular embodiments. It is obvious that those skilled in the art can make modifications and changes to the embodiments without departing from the scope of the present invention. The embodiments of the present invention have been disclosed for exemplary purposes only, and are not intended to limit the scope of the present invention. The scope of the present invention is determined solely by reference to the claims appended thereto.

[0111]

The above-references series of steps can be performed by software, hardware, or a combination thereof. If the series of steps is performed by software, a program forming the software is installed from a recording medium or via a network onto a computer incorporated into a hardware structure or to a general-purpose computer performing a variety of processes, for example.

[0112]

The program can be recorded beforehand onto one of a hard disk and a read-only memory (ROM) as a recording medium. The program can also be stored (recorded) on a removable recording media temporarily or permanently. The recording media includes a floppy disk, a compact disk read-only memory (CD-ROM), a magneto-optic (MO) disk, a digital versatile disk (DVD), a magnetic disk, a semiconductor memory, etc. Such a removable medium can be supplied in package software.

[0113]

The program can be installed from the removable recording medium to the computer. The program can be transmitted in a wireless fashion to the computer from a download site. The program can also be transmitted in a wired fashion via a network such as one of a LAN (local area network) and the Internet. The program is then received by the computer and installed onto a recording medium such as a hard disk in the computer.

[0114]

The process steps discussed in this specification are sequentially performed in the time series order as stated. Alternatively, the steps may be performed in parallel or separately. In this specification, the system refers to a logical system composed of a plurality of apparatuses, and the elements of each apparatus are not necessarily contained

in the same casing.

[Industrial Applicability]

[0115]

In accordance with embodiments of the present invention, the main operating system (OS) executing the interrupt process is set in the system in which a plurality of OS's concurrently run. With the main OS performing interrupt control process, the mask time of the entire system is reduced, interrupt response is improved, and data processing is efficiently performed.

[0116]

In accordance with embodiments of the present invention, the main OS executing the interrupt process control is set, and the right to set interrupt mask is not handed over to the sub OS other than the main OS. The sub OS notifies the main OS whether the sub OS is in the interrupt enabled state or the interrupt disabled state. The main OS controls interrupt mask of the sub OS. This arrangement prevents the sub OS from reserving a required interrupt process due to own mask control. In accordance with the intention of the main OS, all interrupt process is controlled. Required interrupt processes are thus performed with priority.

[0117]

In accordance with embodiments of the present invention, the sub OS interrupt vector management unit is set in the

main OS. The main OS generally manages the interrupt vector area of the sub OS. Unlike interrupt vector management individually performed by OS's, the interrupt vector is shared, and memory area is reduced.